ragtop: Complex Derivatives Pricing

Brian K. Boonstra

First Version: May 20, 2016 This Version: 2019-05-20

Contents

Introduction

ragtop prices equity derivatives on variants of the famous Black-Scholes model, with special attention paid to the case of American and European exercise options and to convertible bonds. Convertible bonds are one of the few types of derivative securities straddling asset classes, and whose valuation must be linked to reasonable models of *multiple* asset types, involving equity valuations, fixed income and sometimes foreign exchange.

A **convertible bond** is similar to a corporate bond[1], promising coupons and notional payments at some known set of future dates, but with a twist. The bond holder, who has effectively lent money to the issuer, can choose to *convert* the bond into equity (subject to some restrictions), in a varying amount known as the *conversion value*. The bond value therefore depends on three major processes:

- *equity* value affecting conversion value
- *default* of the issuer wiping out equity, coupons and notional
- *interest rates* affecting the discounted value of future coupons and notional

Of these processes, the changes in equity value are most important, followed closely by issuer default[2]. We perform derivative pricing and calibration based on simply linked models of equities and corporate defaults.

Stochastic Model

Our basic stochastic model links equity values S_t with *hazard rate* or default intensity h

$$\frac{dS_t}{S_t} = (r + h - q)dt + \sigma dZ - dJ$$

and can be converted to a PDE satisfied by any derivative of S that lacks cashflows

$$\frac{\partial V}{\partial t} - rV + h(\delta - V) + (r - q + h)\, S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = 0.$$

ragtop numerically integrates this PDE using an implicit scheme, forming solutions $v_n^{(m)}$ on a grid of times $t^{(m)}, m = 0, \ldots, M$ and stock prices $S_n, n = -N, \ldots, N$. The present value of our derivative is represented by the entry $v_0^{(M)}$.

[1] A standard corporate bond is often called a *straight* bond

[2] One might wonder how a fixed-income security could be relatively insensitive to stochastic interest rates. Most convertibles are issued in countries with stable economies, so the rates are generally far less variable and far smaller than the credit spreads of the bond issuers.

For further details, please see the technical paper.

Option Market Data

We have included some option market data in **ragtop**, consisting of a set of several hundred option details for Tesla Motor in April 2016. This includes an underlying price

```
TSLAMarket$S0
```

```
[1] 241.8
```

A set of risk-free rates

```
knitr::kable(TSLAMarket$risk_free_rates, digits=3, row.names = F)
```

time	rate
0.050	0.004
0.127	0.005
0.376	0.008
0.625	0.010
0.721	0.010
1.718	0.010
5.000	0.020

and some option price data, excerpted here:

callput	K	time	mid	bid	ask	spread
1	140	0.127	101.775	100.05	103.50	3.45
-1	300	0.127	61.200	60.00	62.40	2.40
1	360	0.376	1.745	1.57	1.92	0.35
1	185	0.625	64.750	63.80	65.70	1.90
-1	440	0.625	205.225	203.45	207.00	3.55
-1	55	1.718	2.635	2.38	2.89	0.51

Pricing Options

Tesla does not pay any dividends, so we can evaluate calls using the Black Scholes model

```
$Price
[1] 69.47563

$Delta
[1] 0.8113372

$Vega
[1] 51.65849
```

or better yet find the implied Black-Scholes volatility

```
implied_volatility(option_price = TSLAMarket$options[400,'ask'],
                   S0 = TSLAMarket$S0,
                   callput = TSLAMarket$options[400,'callput'],
                   K=TSLAMarket$options[400,'K'],
                   r = 0.005,
                   time = TSLAMarket$options[400,'time'])
```

```
[1] 0.4151244
```

For puts, we need to use a pricing algorithm that accounts for early exercise. **ragtop** uses a control variate scheme on top of an implicit PDE solver to achieve reasonable performance

```
american(
       callput = TSLAMarket$options[400,'callput'],
       S0 = TSLAMarket$S0,
       K=TSLAMarket$options[400,'K'],
       const_short_rate = 0.005,
       time = TSLAMarket$options[400,'time'])
```

```
A360_137_2
  4.149565
```

This is also the underlying scheme for implied volatility

```
american_implied_volatility(option_price = TSLAMarket$options[400,'ask'],
     S0 = TSLAMarket$S0,
     callput = TSLAMarket$options[400,'callput'],
     K=TSLAMarket$options[400,'K'],
     const_short_rate = 0.005,
     time = TSLAMarket$options[400,'time'])
```

```
[1] 0.415125
```

Including Default Intensities

We can correct for constant intensities of default with parameters of convenience, both for European exercise

```
implied_volatility(option_price = 17,
                   S0 = 250, callput = CALL,  K=245,
                   r = 0.005, time = 2,
                   const_default_intensity = 0.03)
```

```
[1] NA
```

and for American exercise

```
american_implied_volatility(option_price = 19.1,
     S0 = 223.17, callput = PUT, K=220,
     const_short_rate = 0.005, time = 1.45,
     const_default_intensity = 0.0200)
```

```
[1] 0.1701377
```

Including Term Structures

If we have full term structures, we can use the associated parameters of inconvenience

```
## Dividends
divs = data.frame(time=seq(from=0.11, to=2, by=0.25),
                  fixed=seq(1.5, 1, length.out=8),
                  proportional = seq(1, 1.5, length.out=8))

## Interest rates
disct_fcn = ragtop::spot_to_df_fcn(
  data.frame(time=c(1, 5, 10, 15),
```

```
             rate=c(0.01, 0.02, 0.03, 0.05))
)

## Default intensity
surv_prob_fcn = function(T, t, ...) {
  exp(-0.07 * (T - t)) }

## Variance cumulation / volatility term structure
vc = variance_cumulation_from_vols(
   data.frame(time=c(0.1,2,3),
              volatility=c(0.2,0.5,1.2)))
paste0("Cumulated variance to 18 months is ", vc(1.5, 0))
```

```
[1] "Cumulated variance to 18 months is 0.369473684210526"
```

to modify our estimates accordingly, including on vanilla option prices

```
black_scholes_on_term_structures(
   callput=TSLAMarket$options[500,'callput'],
   S0=TSLAMarket$S0,
   K=TSLAMarket$options[500,'K'],
   discount_factor_fcn=disct_fcn,
   time=TSLAMarket$options[500,'time'],
   survival_probability_fcn=surv_prob_fcn,
   variance_cumulation_fcn=vc,
   dividends=divs)
```

```
$Price
[1] 68.04663

$Delta
[1] 0.8289907

$Vega
[1] 47.06523
```

American exercise option prices

```
american(
    callput = TSLAMarket$options[400,'callput'],
    S0 = TSLAMarket$S0,
    K=TSLAMarket$options[400,'K'],
    discount_factor_fcn=disct_fcn,
    time = TSLAMarket$options[400,'time'],
    survival_probability_fcn=surv_prob_fcn,
    variance_cumulation_fcn=vc,
    dividends=divs)
```

```
A360_137_2
  2.894296
```

and of course volatilities of European exercise options

```
implied_volatility_with_term_struct(
   option_price = TSLAMarket$options[400,'ask'],
   S0 = TSLAMarket$S0,
   callput = TSLAMarket$options[400,'callput'],
```

```
  K=TSLAMarket$options[400,'K'],
  discount_factor_fcn=disct_fcn,
  time = TSLAMarket$options[400,'time'],
  survival_probability_fcn=surv_prob_fcn,
  dividends=divs)
```

```
[1] 0.4112688
```

as well as American exercise options

```
american_implied_volatility(
    option_price=TSLAMarket$options[400,'ask'],
    callput = TSLAMarket$options[400,'callput'],
    S0 = TSLAMarket$S0,
    K=TSLAMarket$options[400,'K'],
    discount_factor_fcn=disct_fcn,
    time = TSLAMarket$options[400,'time'],
    survival_probability_fcn=surv_prob_fcn,
    dividends=divs)
```

```
[1] 0.4090605
```

Fitting Term Structures of Volatility

Let's say we have a some favored picture of our default intensity as a function of stock price and time.

```
def_ints_fcn = function(t, S, ...){
  0.09+0.01*(S0/S)^1.5
  }
```

Let's further postulate a set of financial instruments with known prices. If those instruments all have different maturities, then (generically) there is a unique piecewise constant volatility term structure that will reproduce those prices.

```
options_df = TSLAMarket$options
S0 = TSLAMarket$S0
make_option = function(x) {
  if (x['callput']>0) cp='C' else cp='P'
  ragtop::AmericanOption(callput=x['callput'], strike=x['K'], maturity=x['time'],
                         name=paste(cp,x['K'],as.integer(100*x['time']), sep='_'))
}
atm_put_price = max(options_df$K[options_df$K<=S0])
atm_put_ix = ((options_df$K==atm_put_price) & (options_df$callput==PUT)
              & (options_df$time>1/12))
atm_puts = apply(options_df[atm_put_ix,], 1, make_option)

atm_put_prices = options_df$mid[atm_put_ix]
knitr::kable(options_df[atm_put_ix,], digits=3, row.names = F)
```

callput	K	time	mid	bid	ask	spread
-1	240	0.127	15.925	15.70	16.15	0.45
-1	240	0.376	27.100	26.75	27.45	0.70
-1	240	0.625	35.425	34.60	36.25	1.65
-1	240	0.721	37.400	36.90	37.90	1.00
-1	240	1.718	58.075	56.15	60.00	3.85

Once we have those instruments, we can successively fit our volatility term structure to longer-and-longer dated securities. This is done for us in the `fit_variance_cumulation` function

```
vcm = fit_variance_cumulation(S0, eq_options=atm_puts,
       mid_prices=atm_put_prices,
       spreads=0.01*atm_put_prices,
       use_impvol=TRUE,
       discount_factor_fcn=disct_fcn,
       default_intensity_fcn = def_ints_fcn,
       num_time_steps=100)
vcm$volatilities
```

```
[1] 0.4531763 0.4107467 0.3973169 0.3828557 0.3466839
```

Fitting Default Intensity

True default intensities are unobservable, even retrospectively, so it is nearly impossible to form an historical time series of them. One may try by using credit spreads, but even then an historical calibration may be fairly unreliable for extension into the future. Our model requires those future default intensities, so choosing a reasonable functional form requires close attention. Our choice will generally arise from fitting to available market information.

Since a full fit of the model requires calibration of both variance cumulation and default intensity, our optimization algorithm must work in two phases. It begins with a functional form of the user's choice, such as

$$h(S) = h_0 \left(s + (1-s) \left(\frac{S_0}{S} \right)^p \right)$$

and then takes a set of options, such as these

callput	K	time	mid	bid	ask	spread
-1	210	0.625	21.325	21.00	21.65	0.65
-1	220	0.625	25.475	24.35	26.60	2.25
-1	210	0.721	23.325	22.95	23.70	0.75
-1	220	0.721	27.575	27.05	28.10	1.05
-1	210	1.718	42.100	41.30	42.90	1.60
-1	220	1.718	46.925	45.80	48.05	2.25

and tries to match their prices with the best possible choice of p and s, while still matching the volatility term structure as required above.

To that end, we define an objective (or penalty) function comparing instrument prices P_i to model prices $\tilde{P}_i$

$$\pi(p,s) = \sum \left(\tilde{P}_i - \tilde{P}_i \right)^2$$

```
h0 = 0.05
fit_penalty = function(p, s) {
def_intens_f = function(t,S,...) {h0 * (s + (1-s) * (S0/S)^p)}
varnce = fit_variance_cumulation(
            S0, eq_options=atm_puts,
            mid_prices=atm_put_prices,
            spreads=0.01*atm_put_prices,
            use_impvol=TRUE,
            default_intensity_fcn = def_intens_f,
```

```
              discount_factor_fcn=disct_fcn,
              num_time_steps=100)
  pvs_list = find_present_value(
              S0=S0, instruments=fit_targets,
              default_intensity_fcn=def_intens_f,
              variance_cumulation_fcn=varnce$cumulation_function,
              discount_factor_fcn=disct_fcn,
              num_time_steps=45)
  pvs = as.numeric(pvs_list)
  pensum = sum((fit_target_prices - pvs)^2)
  pensum
}
fit_penalty(1, 0.5)
```

```
[1] 7.918304
```

We can apply our favorite optimizer to this penalty function in order to optimize the model parameters. **ragtop** includes one such fitting algorithm in `fit_to_option_market` and its simpler but less featureful companion `fit_to_option_market_df`. Let's say we have decided $h_0 = 0.04$, $s = 0.75$ and $p = 5/2$. We may have some other instrument we wish to price consistent with this calibration, such as a convertible bond

```
cb = ragtop::ConvertibleBond(
  maturity=2.87, conversion_ratio=2.7788, notional=1000,
  coupons=data.frame(payment_time=seq(2.8,0, by=-0.25),
                     payment_size=1000*0.0025/4),
  discount_factor_fcn = disct_fcn,
  name='CBond'
)

s = 0.75
h0 = 0.04
p = 2.5
```

of course we then need to make sure we have set up the associated variance accumulator function

```
calibrated_intensity_f = function(t, S, ...){
  0.03+0.01*(S0/S)^1.5
}

calib_varnce = fit_variance_cumulation(
              S0, eq_options=atm_puts,
              mid_prices=atm_put_prices,
              spreads=0.01*atm_put_prices,
              use_impvol=TRUE,
              default_intensity_fcn = calibrated_intensity_f,
              discount_factor_fcn=disct_fcn,
              num_time_steps=100)
calib_varnce$volatilities
```

```
[1] 0.4802569 0.4572752 0.4574770 0.4483994 0.4478110
```

Now we simply need to run our pricing algorithm

```
cb_value = form_present_value_grid(
  S0=S0, grid_center=S0,
  instruments=list(Convertible=cb),
  num_time_steps=250,
```

```
  default_intensity_fcn=calibrated_intensity_f,
  discount_factor_fcn = disct_fcn,
  variance_cumulation_fcn=calib_varnce$cumulation_function)
```

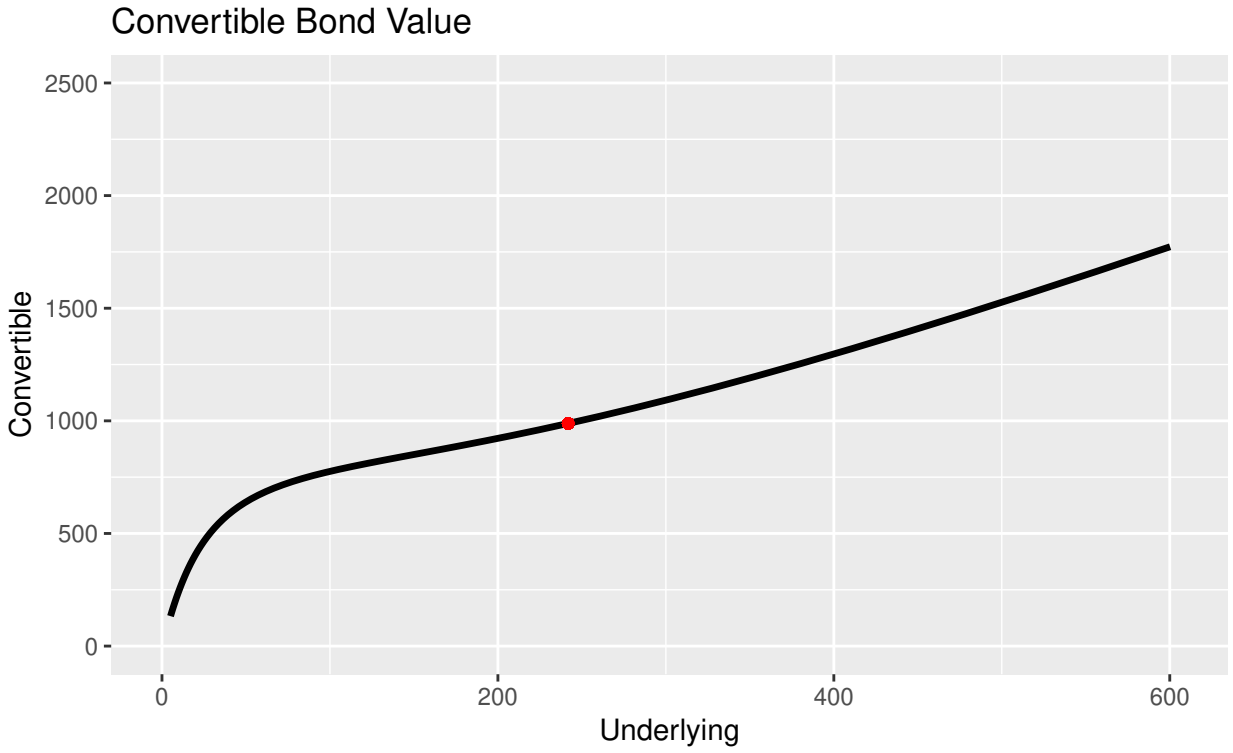
For convenience, **ragtop** also exposes the full grid from its solver, allowing us to calculate delta and gamma

```
cbprices = ragtop::form_present_value_grid(
  S0=S0, grid_center=S0,
  instruments=list(Convertible=cb),
  num_time_steps=250,
  default_intensity_fcn=calibrated_intensity_f,
  discount_factor_fcn = disct_fcn,
  variance_cumulation_fcn=calib_varnce$cumulation_function,
  std_devs_width=5)
cbgrid = na.omit(as.data.frame(cbprices))

present_value_interp = splinefun(
  x=cbgrid[,"Underlying"],
  y=cbgrid[,"Convertible"])

delta = present_value_interp(S0, deriv=1)
delta
```

```
[1] 1.663418
```

or to make a plot

```
present_value = present_value_interp(S0)
cbplot = ( ggplot(cbgrid,
                  aes(x=Underlying,y=Convertible)) +
             geom_line(size=1.2) +
             scale_x_continuous(limits=c(0,2.5*S0)) +
             scale_y_continuous(limits=c(0,2.5*cb$notional)) +
             geom_point(aes(x=S0,y=present_value), color="red") +
             labs(title="Convertible Bond Value")
)
cbplot
```



Daycount Conventions

For dealing with daycount conventions (Act/360, Act/Act, 30/360 and many more), `ragtop` provides no facilities directly. These computations can be outsourced to `quantmod`, though the `BondValuation` package is both lighter in footprint and higher in accuracy. To that end, we can use the helper function `detail_from_AnnivDates()`

```
twitter_bv = BondValuation::AnnivDates(
    Em=as.Date('2018-06-11'),    # Issue date
    Mat=as.Date('2024-06-15'),
    CpY=2,
    FIPD=as.Date('2018-12-15'),   # First coupon
    FIAD=as.Date('2018-06-15'),   # Beginning of first coupon accrual
    RV=1000,   # Notional
    Coup=0.25,
    DCC=which(BondValuation::List.DCC$DCC.Name=='30/360'),  # 30/360 daycount convention
    EOM=0
  )

twitter_specs = ragtop::detail_from_AnnivDates(
    twitter_bv,
    as_of=as.Date('2018-02-15')
  )

twtr_cb = ragtop::ConvertibleBond(
    maturity=twitter_specs$maturity,
    conversion_ratio=17.5001,
    notional=twitter_specs$notional,
    coupons=twitter_specs$coupons,
    discount_factor_fcn = disct_fcn,
```

```
    name='TwitterConvertWithGreenshoe'
  )

pvs = ragtop::find_present_value(
    S0=33.06,
    num_time_steps=200,
    instruments=list(TWTR=twtr_cb),
    const_volatility=0.47,
    const_default_intensity=0.01,
    discount_factor_fcn=disct_fcn,
  )

paste("Twitter bond value is", pvs$TWTR)
```

```
[1] "Twitter bond value is 1031.83208503188"
```

References